

FINAL PROGRESS REPORT
ON THE PHYSICAL REALIZATION OF AN ELECTRONIC COMPUTING INSTRUMENT

by

Herman H. Goldstine

James H. Pomerene

Charles V. L. Smith

THE INSTITUTE FOR ADVANCED STUDY

ELECTRONIC COMPUTER PROJECT

January 1954

PREFACE

This report has been prepared under the terms of Contracts W-36-034-ORD-7481 and DA-36-034-ORD-19 (Project No. TB3-0007 F) between the Research and Development Service, U. S. Army Ordnance Corps and the Institute for Advanced Study. It is a final report on the latter contract, covering the period up to 1 July 1952.

This report is issued in two parts: Part I (text) and Part II (drawings). Part II is separate from this volume and comprises the complete circuit drawings for the completed machine.

Certain accessory devices, notably a magnetic drum and an IBM input-output system, were added in the period subsequent to 1 July 1952. These will be described in a following report.

John von Neumann
Project Director

Institute for Advanced Study

TABLE OF CONTENTS

Part I

Preface

Chassis List

Drawings List

Figures List

I. Mathematical Aspects	1
II. The Orders	22
0.1 The Plus Clear Order	22
0.2 The Plus Hold Order	23
0.3 The Minus Clear Order	24
0.4 The Minus Hold Order	24
0.5 The Plus Absolute Clear Order	25
0.6 The Plus Absolute Hold Order	25
0.7 The Minus Absolute Clear Order	26
0.8 The Minus Absolute Hold Order	26
0.9 The Multiply No-Round Off Order	27
0.10 The Multiply Round Off Order	28
0.11 The Division Order	28
0.12 The Load RII Order	29
0.13 The Store Order	30
0.14 The Store Clear Order	30
0.15-0.16 The Unconditional Transfer Orders	31
0.17-0.18 The Conditional Transfer Orders	32
0.19 The Quick Sum Order	33
0.20 Right Shift, No Round Off Order	34
0.21 Right Shift, Round Off Order	35
0.22 Left Shift Order	35
0.23 The R_2 to R_1 Order	36
0.24-0.25 IBM and Drum Priming Order	37
0.26 IBM Input to Memory Order	38
0.27 IBM Output to Memory Order	39
0.28 Drum Input to Memory Order	39
0.29 Drum Output from Memory Order	40
The digital representation of orders	41

I. MATHEMATICAL ASPECTS

In the succeeding pages of this chapter we shall describe the workings of the principal organs of the machine insofar as they concern the preparation of codes. We assume the reader is familiar with a previous report entitled, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument" (1946) by Burks, Goldstine, and von Neumann; in future references we indicate this report by PD. In this chapter we discuss those features of the arithmetic part of the machine which are relevant from a mathematical point of view.

In a consideration of the Arithmetic Organ one is naturally led first to discuss the number system employed. In spite of a long standing tradition in favor of the decimal system we were led both by logical and engineering considerations to employ the binary system. Since the control portions of the machine are carrying out purely logical functions and since logics are best expressed as binary operations the reasons for a binary representation, at least, of the orders for the machine are evident. On the engineering side the components out of which the machine is constructed are again binary in nature: The "flip-flop" is fundamentally a binary device; the "gate" is also; and the process of storing charge in the dielectric face or screen of the cathode ray tube used in the Memory is again of this same character. Hence, if one contemplates employing the decimal system, one is forced to a binary coding of the decimal system, each decimal digit being represented by a tetrad of binary digits. Thus a precision of 10 decimal digits would require

40 binary digits. But in a true binary representation about 33 digits suffice to achieve a precision of 10^{10} . Thus one is led to use Memory space -- recall that this is the most "expensive" portion of the instrument -- wastefully. It will also be seen as the discussion proceeds that the arithmetic portions of the machine are much simpler logically and hence engineering-wise in the binary system than in the decimal one.

To illustrate this latter point consider the problem of multiplication. In the binary system the product of a number x by a binary digit is either x or null according as the digit is 1 or 0. In the decimal system, on the other hand, there are ten possible values for the product of a digit by x , $0.x$, $1.x$, ..., $9.x$. Thus decimal multiplication is fundamentally a more complex operation than is the binary one and this will be expressed in a decimal instrument either by a circuit complication or by the multiplication being slower. Similar remarks can be made about the other arithmetic processes.

It is often argued that notwithstanding these complications the decimal system is easier from the human point of view. Our machine, however, is such that data may be introduced either binarily or decimally and can be withdrawn in the same fashion if desired, and this without any circuitry. The conversions are trivially handled by extremely simple codes.

It is perhaps well to give at this point some details on the method of introducing data into the machine to enable the reader to develop gradually a feeling for the overall economy of our establishment. Each piece of information is introduced as an aggregate of 10 quantities in

the hexadecimal system. In this number system there are 15 integers $\bar{0}$, ..., $\bar{9}$, $\bar{10}$, $\bar{11}$, $\bar{12}$, $\bar{13}$, $\bar{14}$, $\bar{15}$ which we call 0, ..., 9, A, B, C, D, E, F. Thus the first 10 of these integers are exactly the decimal integers, so that a decimal quantity introduced into the machine is given its familiar and usual form. A binary number or order -- these are in binary form as will be explained in the chapter on the code -- is expressed as 10 tetrads of binary digits, i.e. as 10 hexadecimal integers.

The decision as to whether a given quantity is to be treated by the machine as the decimal representation of a given number or as the hexadecimal representation of a binary number is left to the coder. I.e., he knows which of the data he has introduced is decimal and must be converted by the machine into a binary form and which is already binary. This decision places no more burden on the coder than does that one which requires him to know which data are orders and which are numbers. Indeed, the two problems are quite intimately related. Generally, in coding a given problem it is the practice to place in a block of consecutive positions the decimal information. This makes the conversion of these numbers into their binary form a simple inductive procedure determined only by the number of places desired and the locations of the initial and terminal quantities.

We leave this subject for the present and return to it later after we have described the orders themselves.

The Arithmetic Organ is a 40-fold aggregate of binary units. We use the first of these to record the sign digit of a number and the remaining 39 for digital information. Thus each "word", i.e. aggregate of

40 binary digits, viewed as a binary number has a precision of $2^{-39} \sim 10^{-11.7}$. We have chosen to fix our binary point immediately to the left of the first digit of numerical material, i.e. the binary point is fixed immediately after the sign digit. Thus the digits -- apart from the sign -- have positional values $2^{-1}, 2^{-2}, \dots, 2^{-39}$. As a matter of fact, as far as our Adder is concerned the sign is treated as a binary digit with positional value 2^0 .

Before proceeding from this point it is well to discuss our treatment of negative numbers in the machine since this has bearing on the character of the Arithmetic Organ. To do this we say first a word about our Adder. If one regards our numbers $x = (x_0, x_1, \dots, x_{39})$ as 40-digit quantities $x_0 \cdot 2^0 + x_1 \cdot 2^{-1} + \dots + x_{39} \cdot 2^{-39}$, then our Adder as far as digit-adding and carrying mechanisms are concerned functions identically in all places with one exception: If a carry proceeds from the left-most digit, it is "lost" (cf., however, our discussion below of the division operation). This means clearly that the augend and addend, both of which lie between 0 and 2 have produced a sum greater than 2 will omit the 2. This is, of course, nothing other than a statement that the Adder functions modulo 2.

In this sense all numbers represented in the machine can be viewed as being modulo 2. We have used this fact to determine our representation of negative numbers. If x is an arbitrary real number, then there is exactly one number \bar{x} between 0 and 2 with which it agrees modulo 2, i.e. for each x there is a unique \bar{x} such that $0 \leq \bar{x} < 2$ and $x \equiv \bar{x} \pmod{2}$. This fact fixes our representation of negative numbers.

We agree always to deal with numbers x for which $-1 \leq x < 1$. Now the \bar{x} associated with x is x if $x \geq 0$; thus, $0 \leq \bar{x} < 1$ in this case we represent x by the digitalized form of \bar{x} . It clearly has x_0 , its sign digit +, i.e. 0. If $x < 0$, then $x = \bar{x} + 2$ and we have $1 \leq \bar{x} < 2$, i.e. the left-most digit of \bar{x} is 1, i.e. -. Thus we always represent a number x by the digitalized form of \bar{x} and have the convention that + is 0 and - is 1 with the left-most digit being the sign.

In closing this discussion we mention the relation of our representation of negative numbers with that of "complementation". Consider a negative number x with $-1 \leq x < 0$ and let $y = -x$. Then $0 < y \leq 1$. As we said above we digitalize x by representing it as $x + 2 = 2 - y = 1 + (1 - y)$. Then the left-most digit of this representation is, correctly, 1 and the remaining digits are those of the complement of $y = |x|$. This is what is frequently called the representation by complementation of negative numbers.

The Arithmetic Organ proper contains the following principal units: 3 Registers of 40 digits each (cf. however, below for an exception to this), an Adder, various sets of gates whose functions will be made clear in what follows, and a Control Unit to supervise the performance of the various Arithmetic orders. In the accompanying figure we show schematically the interrelations between some of these and in later figures we show more details.

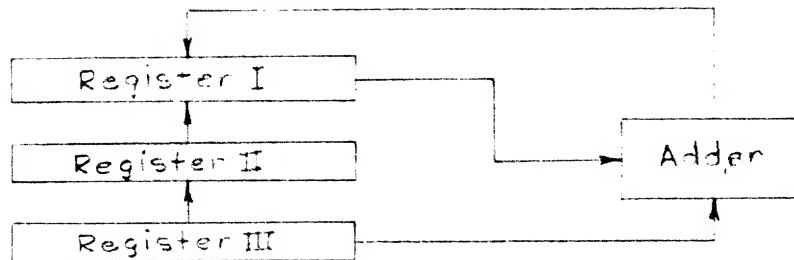


Fig. I. 1.

As indicated in the figure the inputs (40-fold in each case) to the Adder are from Registers I and III -- we shall use the symbols RI and RIII in the future -- and the output of this unit is stored back in RI. Again information in RIII can be communicated to RII and hence to RI without proceeding through the Adder. In an addition the augend is originally in RI and the addend in RIII, the sum being placed in RI at the completion of the operation.

To make clear the subtraction we must make mention of a unit called the Complement Gate Chassis which intervenes between RIII and the Adder, as indicated in the figure below.

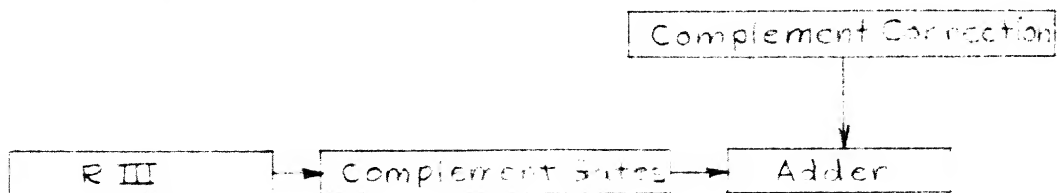


Fig. I. 2.

This chassis permits one of three modes of communication between RIII and the Adder. If a number x is stored in RIII, then either the Complement Gates permit x , the complement of x or 0 to enter the Adder; or, to be more precise, if $x = (x_0, x_1, \dots, x_{39})$, then either x or $(1 - x_0, 1 - x_1, \dots, 1 - x_{39})$ or $0 = (0, 0, \dots, 0)$ is permitted to enter the Adder from RIII. If it is the middle case, then the Arithmetic Control also "injects" a digit 2^{-39} into the Adder. This, as we shall see, correctly handles the operation of subtraction.

To form $x - y$ we proceed as follows: the Arithmetic Organ has x in RI, y in RIII and has been instructed to form the difference. It forms, apart from the "complement correction" just mentioned,

$$\sum_{i=0}^{39} (x_i + (1 - y_i)) \cdot 2^{-i} = x + (2 - y) - 2^{-39}.$$
 The complement correction then has the effect of removing this last 2^{-39} and yielding the correct difference.

We leave this discussion of the separate units of the Arithmetic Organ for the moment but with the intention of returning to it shortly.

The multiplication operation is somewhat more delicate in a certain sense than are the addition and subtraction because the procedure based on the modulo 2 fails completely. If one changes one factor, say x , of a product xy by a two, then the new product differs from xy by $2y$ which is not generally an integer multiple of 2 since $-1 \leq y < +1$.

To effect a multiplication we store the multiplier in RII and the multiplicand in RIII. We carry out the process by serially multiplying the entire multiplicand by the digits of the multiplier starting with the least significant one.

The multiplication proper takes place in 39 steps, corresponding to the 39 non-sign digits of the multiplier, together with several "clean-up" operations in addition. We describe all this below. For simplicity we first consider the case in which both the multiplier $x = (x_0, x_1, \dots, x_{39})$ and the multiplicand $y = (y_0, y_1, \dots, y_{39})$, i.e. $x_0 = y_0 = 0$ and hence $0 \leq x < 1$, $0 \leq y < 1$.

Assume we have already performed the first $i - 1$ steps of the multiplication involving the multiplication of the multiplicand by the last $i - 1$ digits of the multiplier, $x_{39}, x_{38}, \dots, x_{40-i}$. We describe now the multiplication with the i -th digit, i.e. with x_{40-i} . Assume that RI contains the partial product after the last step, p_{i-1} (for $i = 1, p_0 = 0$). We form

$$2p_i = p_{i-1} + y_k \quad \text{with} \quad y_k = \begin{cases} 0 & \text{for } x_{40-i} = 0 \\ y & \text{for } x_{40-i} = 1. \end{cases}$$

I.e. if $x_{40-i} = 0$ we define p_i as $1/2$ of p_{i-1} and if $x_{40-i} = 1$ as $1/2$ of $(p_{i-1} + y)$. Consider now the sizes of the quantities $2p_i$. For $i = 0$, $0 \leq 2p_i < 2$ (since $p_0 = 0$); now if this is true for $i - 1$, then our displayed definition above makes it also true for i . Thus $2p_i$ lies in the interval $0 \leq 2p_i < 2$ and no carry can arise beyond 2^0 -position.

Thus p_i is formed from $2p_i$ by a right shift with the sign digit made 0. Finally we have

$$\begin{aligned} p_{39} &= 2^{-1}(2^{-1}(2^{-1}(\dots(2^{-1}x_{39}y + x_{38}y)\dots) + x_1y) = \\ &= \sum_{i=1}^{39} 2^{-i} x_i y = xy, \end{aligned}$$

i.e. we have our correct product. We describe later how we achieve this in the Arithmetic Organ. At the moment, however, we turn instead to the

other possible cases, namely: $x < 0, y \geq 0, x < 0, y < 0, x \geq 0, y < 0$.

We pass now to these cases and describe how they are performed.

If $x < 0$, then it is represented in the machine as $x + 2$. Thus for $x < 0, y \geq 0$ the procedure we have just described would form not xy but $xy + 2y$; for $x < 0, y < 0$ it would form $xy + 2x + 2y + 4$; for $x \geq 0, y < 0$ it would form $xy + 2x$. Hence, correction terms $2x, 2y$ or both would be needed. As we shall see later these corrections would be quite awkward for us to perform, particularly the correction $2x$ since we in fact lose the digits of the multiplier as they are no longer needed. The reason for this will become apparent in the next section.

Our procedure is this: First let us assume that the corrections necessitated by $y < 0$ have been disposed of and permit y to be either ≤ 0 or > 0 . We focus attention now on $x < 0$.

We disregard the sign digit of x and act as if it were 0. Then x is replaced by $x^1 = x - 1$ but since $-1 \leq x < 0, x^1$ will act as if it were $(x - 1) + 2$. Hence our procedure for multiplication will produce $x^1 y = (x + 1)y = xy + y$. We therefore need a final correction in this case of $-y$ at the end of the process. Thus in the cases $x < 0$ we proceed through the 39 steps described earlier and thereby form $xy + y$ and then we must perform another step to subtract out the multiplicand y .

Having disposed of the difficulties that arise when $x < 0$, we may now assume $x > 0$ and consider the one remaining case, namely $y < 0$.

Suppose this time that we ignore completely the sign digit of y , or rather that we replace it by 0. Then if $y^1 = y - 1$, we have as before $xy^1 = x(y + 1) = xy + x$ and a correction $-x$ is needed. Since, however,

we do not have x , the multiplier, available at the end of the multiplication we must find a means of applying this correction as the first 39 steps proceed.

We proceed in this fashion: when we examine the digit x_{39-i} of the multiplier, we normally add into the partial product p_i the number y if $x_{39-i} = 1$ and 0 otherwise. Let us now modify this procedure as follows:

$$2p_i = p_{i-1} + \hat{y}_i \quad \text{with} \quad \hat{y}_i = \begin{cases} 1 & \text{for } x_{40-i} = 0 \\ y^1 & \text{for } x_{40-i} = 1. \end{cases}$$

As before $0 \leq 2p_i < 2$ and no carries can proceed beyond the 2^0 -position. Let us see now what result we have produced by this procedure.

$$\begin{aligned} p_{39} &= 2^{-1}(2^{-1}(2^{-1}(\dots(2^{-1}x_{39}y^1 + 2^{-1}(1-x_{39}) + x_{38}y^1 + (1-x_{38}))\dots) + \\ &\quad + x_2y^1 + (1-x_2)) + x_1y^1 + (1-x_1) = \\ &= \sum_{i=1}^{39} 2^{-i} x_i y^1 + \sum_{i=1}^{39} 2^{-i} (1 - x_i) = xy^1 + 1 - 2^{-39} - x = \\ &= x(y+1) + 1 - 2^{-39} - x = xy + (1 - 2^{-39}). \end{aligned}$$

Thus a final correction of $-1 + 2^{-39}$ is necessary. But this correction which is done at the end can be effected modulo 2 and we can correct it by $1 + 2^{-39}$.

We summarize now in a general description covering all four cases.

We return now to our schematic discussion of the Arithmetic Organ. Since we wish to retain the full 78 digits of a product, we have established certain interconnections between RI and RII not yet shown in Figure I.1. Before describing them we must indicate another feature of RI and RII. Each of them is capable not only of receiving 40 digit

numbers and of transmitting them, but also of translating either to the right or left whatever information is stored in them. We discuss the logical implications of these shift facilities later. At the moment we prefer to indicate how this is accomplished, at least in a crude way.

Each of RI and RII is in reality not one but two registers suitably interconnected. Let us consider RI first. It consists of two registers and four sets of 40-fold gates. Let the two registers be denoted by R^I and R_I . One set of gates intervenes between the Adder and R^I . Thus the output of the Adder is stored at least initially in R^I . Two sets of gates allow communication from R^I to R_I and a fourth set allows communication from R_I to R^I .

The set which controls the communication between the Adder and R^I , the so-called Green Gates, is so wired that it makes digital position 2^{-i} of the Adder correspond to 2^{-i} of R^I . One of the two sets controlling the route from R^I to R_I , the so-called Red Gates, makes position 2^{-i} of R^I correspond to $2^{-(i-1)}$ of R_I ; the other set, the so-called Black Gate, makes 2^{-i} of R^I correspond to $2^{-(i+1)}$ of R_I . The fourth set, the so-called Yellow Gate, from R_I to R^I makes 2^{-i} of R_I correspond to 2^{-i} of R^I . We indicate this below in Figure I.3.

A similar arrangement obtains with respect to RII. The structure of RIII is, however, simpler since it is not called upon to perform shifting functions as are RI and RII.

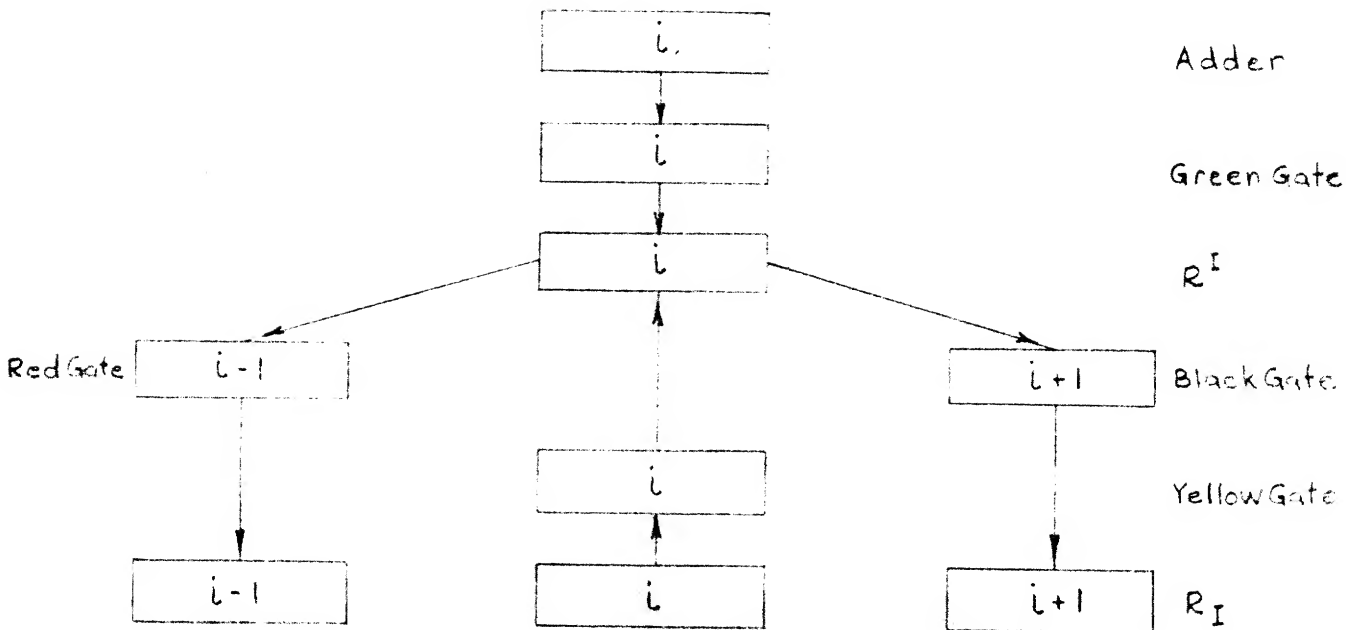


Figure I.3.

We can now describe in somewhat more detail the operations previously alluded to such as the right and left shifts, the transfer into R_I , the addition, the subtraction, and at least part of the multiplication.

Consider first a number in R_I -- in both R_I and R_{II} the units R^I , R^{II} serve only as transient storage positions; all storage for more than a few microseconds is in R_I , R_{II} -- which we desire to shift right (left). The Arithmetic Control routes the information first to R^I via the Yellow Gate set, then back to R_I via the Black (Red) Gate set. (We must describe later treatment of the sign digit.)

Next consider a number arriving in R_I from the Adder. It is transferred to R^I via the Green Gate set, then to R_I via the Red set -- note this apparently causes the information in 2^0 -position of the Adder

to be lost. Actually it does not because R_I has a 2^{+1} position for precisely this reason; it is then sent back to R_I^I via the Yellow set and finally back to R_I via the Black set. Note that it is now correctly positioned, i.e. the content of 2^{-1} -position of R_I is that of 2^{-1} -position of the Adder.

In the next figure we indicate the connections between R_I and the Adder, suppressing the gate sets.

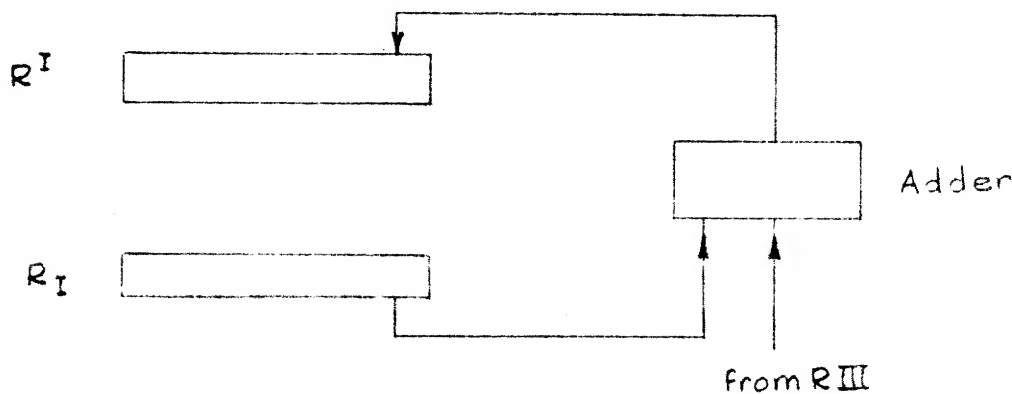


Figure I.4.

Thus in an addition it is the quantity stored in R_I that is added to that in R_{III} , the result being placed temporarily in R_I^I .

It should also be clear from what has been said how the right and left shifts are performed. We must return later to describe certain details of these operations, such as the treatment of the sign digit in the right shift and of the right-most digit in a left shift.

Before doing this, however, we must first describe two more interconnections between R_I and R_{II} . All interconnections described previously have been 40-fold but the one we now describe differs in that they are 1-fold. Specifically the right-most stage of R_I is connected to the left-most one of R_{II} in such a fashion that the route is

from RI to RII but not back again. This connection is provided so that whenever a right shift occurs the digits being shifted out of RI are stored in RII. We make this connection quite specific in Figure I.5 below.

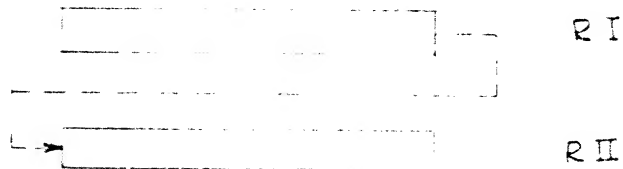


Figure I.5.

To provide for the comparable situation when a left shift occurs the 2^0 stage of RI is connected to the 2^{-39} stage of RII, as in Figure I.6 below.

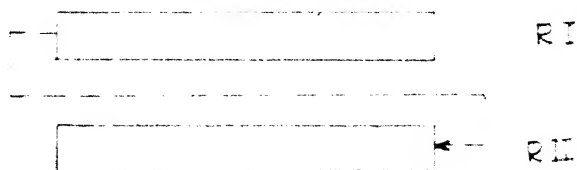


Figure I.6.

We are now able to proceed further with the details of the multiplication. The multiplier is initially placed in RII. (This must be done prior to the multiplication order, c.f. 0.9 below.) Then when the multiplication is initiated the multiplicand is in R^{III} . An observation post exists at stage 39 of RII which examines whether the digit therein is 0 or 1 and acts accordingly, i.e. it does not or does

add the multiplicand into RI in case both multiplier and multiplicand are positive. We discuss below the exact details in all cases. Then a right shift of one is performed. Thus three things occur of relevance: first, the partial product in RI is properly positioned for the next step; second, the digit of the multiplier last examined has been lost and the next relevant, i.e. the now currently relevant, is available at the inspection station; third, the least significant digit of the partial product has now been shifted into RII, into the leading stage. This procedure is carried on for the 39 steps required at which time the 39 most significant digits of the product appear in RI and the 39 least significant ones in RII.

The addition operation is performed in this fashion: We assume the augend is now in RI, specifically in R_I , and the addend is in R^{III} . The Complement Gates are set to pass the addend out and the sum is then stored temporarily in R^I . This sum is then put into R_I displaced one to the left with the sign digit in 2^{+1} . Next, the number is transferred back to R^I and thence down to R_I in the correct position. In terms of the various gating operations this means the following: The Green Gates were opened to admit the sum to R^I ; the Red Gates sent it to R_I ; the Yellow Gates sent it back to R^I ; and finally it arrived correctly positioned in R_I via the Black Gates.

The situation for the subtraction differs in one point only; the Complement Gates are opened to pass the complement of the addend and the complement correction is carried out.

In both cases the sign of the sum is now both in 2^{+1} and 2^0 .

The possible addition and subtraction operations performable by the machine are these:

1. The addition (subtraction) of the contents of R^{III} and of R_I .
2. The addition (subtraction) of the contents of R^{III} and of R_I pre-cleared to 0. I.e. the transfer of a number (or its complement) into R_I .
3. The addition (subtraction) of the absolute value of the contents of R^{III} and of R_I .
4. The addition (subtraction) of the absolute value of the contents of R^{III} and of R_I pre-cleared to 0. I.e. the transfer of the modulus of a number (or its complement) into R_I .

To perform the operations involving absolute values the Arithmetic Control is provided with a monitor which decides whether the Complement Gates are to pass the number in R^{III} or its complement according as the instruction requires.

The left shift is performed analogously to that for the right shift but the right-most stage of R_I is made 0.

This is the correct convention to ensure that the left shift is exactly a multiplication by 2 (provided that the result is still in "scale", i.e. is not outside the interval $-1 < x < 1$).

The left shift operation can be performed n times ($1 \leq n \leq 47$) by means of a single order.

The right shift operation is performed in this fashion: The number in R_I is transferred into R^I and is then sent back into R_I displaced one position to the right. Exactly the same procedure is followed in R_{II} .

thus both RI and RII shift together. (There is one exception to this principle in one of the terminal steps of a multiplication but this need not concern us here.)

The information stored in 2^{+1} of RI is therefore shifted into 2^0 . It is also retained in 2^{+1} . If this digit is a 0, the sign of the resulting quantity is 0 and if it is a 1, the sign is 1. But this is exactly the correct convention to ensure that the right shift is exactly a division by 2.

The right shift operation can be performed n times ($1 \leq n \leq 47$) by means of a single order.

This amounts only to an iteration n times of what is described above.

Since RI and RII are interconnected as shown in Figure I.5 above, the information shifted out of RI is transferred into RII; but the material shifted out of RII is lost.

We next discuss the division operation. To make precise what follows we agree that the dividend is x , the divisor is y with $-1 \leq x < 1$, $-1 \leq y < 1$, $|x| < |y|$.

To describe the process we assume that the first $i-1$ steps of the division have been completed and that the first $i-1$ digits q_0, q_1, \dots, q_{i-2} of the quotient Q are in positions $40-i, 41-i, \dots, 39$, respectively. We also assume that y , the divisor, is in R^3 and that the remainder r_{i-1} is in R_1 . We proceed inductively in this fashion:

$$(1) \quad r_i = 2r_{i-1} - (\text{sgn } xy) y p_{i-1}, \quad r_0 = x/2,$$

where

$$(2) \quad p_{i-1} = \begin{cases} 0 & \text{sgn } r_{i-1} \neq \text{sgn } (2r_{i-1} - (\text{sgn } xy) \cdot y) \\ 1 & \text{sgn } r_{i-1} = \text{sgn } (2r_{i-1} - (\text{sgn } xy) \cdot y). \end{cases}$$

$$p_0 = 0$$

We next define q_i as

$$(3) \quad q_i = \begin{cases} p_i & \text{sgn } xy = +1 \\ 1-p_i & \text{sgn } xy = -1 \end{cases}$$

We now show that

$$\text{sgn } r_i = \text{sgn } x, \quad |r_i| < |y|.$$

We prove these inductively. They are evidently true for $i = 1$. We show they are true for $i + 1$ assuming they are true for i . If

$$\text{sgn } r_i = \text{sgn } (2r_i - (\text{sgn } xy) \cdot y)$$

then

$$r_{i+1} = 2r_i - (\text{sgn } xy) \cdot y$$

and

$$\text{sgn } r_{i+1} = \text{sgn } r_i = \text{sgn } x.$$

Next,

$$\begin{aligned} 2r_i - (\text{sgn } xy) \cdot y &= 2 \text{sgn } r_i \cdot |r_i| - \text{sgn } x \cdot \text{sgn } y \cdot y = \\ &= 2 \text{sgn } x \cdot |r_i| - \text{sgn } x \cdot |y| = \text{sgn } x (2|r_i| - |y|). \end{aligned}$$

Thus

$$r_{i+1} = \text{sgn } r_{i+1} \cdot |r_{i+1}| = \text{sgn } x \cdot |r_{i+1}| = \text{sgn } x (2|r_i| - |y|)$$

and

$$|r_{i+1}| = |2r_i| - |y| < 2|y| - |y| = |y|,$$

which completes the induction in this case. In the contrary case

$$r_{i+1} = 2r_i,$$

and

$$\operatorname{sgn} x = \operatorname{sgn} r_i \neq \operatorname{sgn} (2r_i - (\operatorname{sgn} xy) \cdot y) = \operatorname{sgn} x \cdot \operatorname{sgn} (2|r_i| - |y|).$$

Thus

$$\operatorname{sgn} (2|r_i| - |y|) = -1,$$

i.e.

$$|r_{i+1}| = 2|r_i| < |y|,$$

and $\operatorname{sgn} r_{i+1} = \operatorname{sgn} 2r_i = \operatorname{sgn} r_i = \operatorname{sgn} x$, since $2r_i$ is in the machine's number range. Hence we have proved our induction.

We multiply both sides of (1) by 2^{-i+1} and sum for $i = 1, 2, \dots, n$. We find

$$2^{-(n-1)} r_n = 2^1 r_0 - (\operatorname{sgn} xy) \cdot y P,$$

where

$$P = \sum_{i=0}^{n-1} 2^{-i} p_i.$$

Thus

$$(4) \quad x = (\operatorname{sgn} xy) P \cdot y + R$$

where

$$R = 2^{-n+1} r_n$$

since $2^1 r_0 = x$.

If $\operatorname{sgn} xy = +1$, (4) becomes with the help of (3)

$$x = Q y + R,$$

where

$$Q = \sum_{i=0}^{n-1} 2^{-i} q_i = P.$$

If $\operatorname{sgn} xy = -1$, then

$$Q = \sum_{i=0}^{n-1} 2^{-i} q_i = \sum_{i=0}^{n-1} 2^{-i} (1-p_i) = 2 - P = 2^{-n+1},$$

i.e. apart from the term $2^{-(n-1)}$ Q is the complement of P . Thus our quotient is wrong in the last place.

Up to this point we have made no mention of "rounding" procedures. We do not wish in this place to discuss the theoretical background of such procedures. Instead we merely call the reader's attention to such a discussion in a previous report¹⁾ and state the rules we have adapted. In the multiplication operation a digit is added to 2^{-40} and the result truncated after 39 digits after all carries have been completed. In the division operation we perform 39 steps determining the sign and 38 information digits. The 39th such digit is automatically made 1.

We complete our discussion with a discussion of the roles of RI, RII, RIII during the division operation.

At the start of this operation the dividend is in RI, the divisor is in RIII. Although left shifts are to be performed the channel from RI to RII which normally transmits for a left shift is suppressed. Instead the quotient digits are inserted seriatim into position 38 of RII and shifted left. The operation continues until the sign digit of the quotient reaches position 0 of RII. At this time the remainder is in RI.

It remains only to explain how the machine makes the discriminations indicated in (2), (3) above. First we note that in (2) the expression " $\text{sgn } r_{i-1}$ " can be replaced by $\text{sgn } x$. Thus (2) becomes

1) Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, Burks, Goldstine and von Neumann, Pt. I, Vol. I, 1946, pp. 19, ff.

$$(2') \quad p_{i-1} = \begin{cases} 0 & \text{sgn } x \neq \text{sgn } (2r_{i-1} - (\text{sgn } xy) y) \\ 1 & \text{sgn } x = \text{sgn } (2r_{i-1} - (\text{sgn } xy) y). \end{cases}$$

It was not convenient engineering-wise to detect the signum of $2r_{i-1} - (\text{sgn } xy) y$ and therefore a somewhat different quantity was observed. To explain this we suppose for the moment that

$$s = 2r_{i-1} - (\text{sgn } xy) y$$

is expressed not as

$$s = \sigma_0 + \sigma_1/2 + \sigma_1/2^2 + \dots + \sigma_{n-1}/2^{n-1}$$

but as

$$s = \sigma_{-1} + \sigma_0/2 + \sigma_1/2^2 + \dots + \sigma_{n-1}/2^{n-2}.$$

I.e. we regard 0 not as a sign digit but as an arithmetic digit and $_{-1}$ as the sign digit. The convention adopted is now this:

$$(2'') \quad p_{i-1} = \begin{cases} 0 & (1 - \text{sgn } x)/2 = \sigma_{-1} \\ 1 & (1 - \text{sgn } x)/2 \neq \sigma_{-1}. \end{cases}$$

It is not difficult to see that conventions (2') and (2'') are equivalent.

II. THE ORDERS

We proceed now to an explanation of each order in terms of the contents of RI, RII, RIII and of certain other facts relevant to the coder. It is desirable first, however, to mention the digital structure of the orders.

Each order consists of 20 binary digits, the first 10 of which usually specify a Memory location and the second 10 of which specify the operation to be performed. Two orders are grouped together into a single 40 digit word. The Control is so arranged that it first executes the left-hand one of the pair and then the right-hand one. These are referred to as the first and second phases of the order-word, respectively. In what follows we number the digits of an order 0 through 9 for the Memory location and 10 through 19 for the operation.

We now describe the orders.

0.1. THE PLUS CLEAR ORDER.

a) This order may be in either the first or second phase of an order-word.

b) The digits 0-9 (20-29) express the Memory location from which operand is to come.

c) The so-called step-digit, digit 11, may be a 0 or a 1. In either case the order is executed. In the former case the Control is prevented from proceeding to the next order and the machine stops. If after the stop the step digit is changed to a 1, the order is re-done and the machine proceeds normally.

d) At the start of the order the contents of the registers are:

R_1 irrelevant

R_2 irrelevant

R^3 irrelevant

x \cdot b, the addend

e) At the end of the order the contents of the registers and of memory location x are:

R_1 b

R_2 unchanged

R^3 b

x b

0.2. THE PLUS HOLD ORDER.

a), b), c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

R_1 a, the augend

R_2 irrelevant

R^3 irrelevant

x b, the addend.

e) At the end of the order the contents of the registers and x are:

R_1 a + b

R_2 unchanged

R^3 b

x b

0.3. THE MINUS CLEAR ORDER.

a), b), c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

 R_1 irrelevant R_2 irrelevant R^3 irrelevant x b , the subtrahend

e) At the end of the order the contents of the registers and x are:

 R_1 $2 - b$ R_2 irrelevant R^3 b x b

0.4. THE MINUS HOLD ORDER.

a), b), c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

 R_1 a , the minuend R_2 irrelevant R^3 irrelevant x b , the subtrahend

e) At the end of the order the contents of the registers and x are:

 R_1 $a - b$

R_2 unchanged
 R^3 b
 x b

0.5. THE PLUS ABSOLUTE CLEAR ORDER.

a), b), c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

R_1 irrelevant
 R_2 irrelevant
 R^3 irrelevant
 x b, the addend

e) At the end of the order the contents of the registers and x are:

R_1 | b |
 R_2 unchanged
 R^3 b
 x b.

0.6. THE PLUS ABSOLUTE HOLD ORDER.

a), b), c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

R_1 a, the augend
 R_2 irrelevant
 R^3 irrelevant
 x b, the addend

e) At the end of the order the contents of the registers and x are:

$$R_1 \quad a + |b|$$

$$R_2 \quad \text{Unchanged}$$

$$R^3 \quad b$$

$$x \quad b.$$

0.7. THE MINUS ABSOLUTE CLEAR ORDER.

a), b), c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

$$R_1 \quad \text{irrelevant}$$

$$R_2 \quad \text{irrelevant}$$

$$R^3 \quad \text{irrelevant}$$

$$x \quad b, \text{ the subtrahend}$$

e) At the end of the order the contents of the registers and x are:

$$R_1 \quad 2 - |b|$$

$$R_2 \quad \text{unchanged}$$

$$R^3 \quad b$$

$$x \quad b.$$

0.8. THE MINUS ABSOLUTE HOLD ORDER.

a), b), c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

R_1 a, the minuend
 R_2 irrelevant
 R^3 irrelevant
 x b, the subtrahend

e) At the end of the order the contents of the registers and x are:

R_1 a - b
 R_2 unchanged
 R^3 b
 x b.

0.9. THE MULTIPLY NO-ROUND OFF ORDER.

- a) The same as for 0.1.
- b) The digits 0-9 (20-29) express the memory location from which the multiplicand is to come.
- c) The step digit is as in 0.1. The clear digit 18(38) may be a 0 or a 1. In the former case the contents of RI at the start of the order will be added to the first partial product. I.e., if c is in RI at the start and if the desired product is a b, then what is produced in this case is $ab + 2^{-39}d$. In the latter case the contents of RI are cleared to 0 at the start of the multiplication.
- d) At the start of the order the contents of the registers and x are:

R_1 irrelevant if "clear"
 R_2 a, the multiplier
 R^3 irrelevant
 x b, the multiplicand.

e) At the end of the order the contents of the registers and x are:

$$R_1 \quad c_0, c_1, \dots, c_{39}$$

$$R_2 \quad (1-b_0), c_{40}, \dots, c_{78}; \text{ where } ab = \\ = c_0, c_1, \dots, c_{39}, c_{40}, \dots, c_{78} \text{ and } b_0 \text{ is} \\ \text{the sign digit of } b. \text{ (We assume the "clear"} \\ \text{case.)}$$

$$R^3 \quad b$$

$$x \quad b.$$

0.10. THE MULTIPLY ROUND-OFF ORDER.

a), b), c) The same as for 0.9.

d) The same as for 0.9.

e) At the end of the order the contents of the registers and x are:

$$R_1 \quad \delta_0, \delta_1, \delta_2, \dots, \delta_{39}$$

$$R_2 \quad (1-b_0), c_{40} + 1, c_{41}, \dots, c_{78}; \text{ where } \delta_0, \\ \delta_1, \dots, \delta_{39}, c_{40} + 1, \dots, c_{78} - (c_0, c_1, \\ \dots, c_{39}; c_{40}, c_{41}, \dots, c_{78}) = 2^{-40}.$$

$$R^3 \quad b$$

$$x \quad b.$$

0.11. THE DIVISION ORDER.

a) The same as for 0.1.

b) The digits 0-9 (20-29) express the memory location from which the divisor is to come.

c) The same as for 0.1.

d) At the start of the order the contents of the registers and x are:

R_1 N , the dividend

R_2 irrelevant

R^3 irrelevant

x D , the divisor.

e) At the end of the order the contents of the registers and x are:

R_1 $2R$, twice the remainder

R_2 $q_0, q_1, \dots, q_{38}, 1$, cf. Chapter I

R^3 D

x D

0.12. THE LOAD RII ORDER.

a), b) The same as for 0.1.

c) The same as for 0.1. If the clear digit is a 1, then R_1 is pre-cleared to 0.

d) At the start of the order the contents of the registers and x are:

R_1 Irrelevant

R_2 Irrelevant

R^3 Irrelevant

x b

e) At the end of the order the contents of the registers and x are:

R_1 Unchanged if clear digit is 0; 0 if clear digit is 1.

R_2 b
 R^3 b
 x b .

0.13. THE STORE ORDER.

- a) The same as for 0.1.
- b) The digits 0-9 (20-29) express the memory location into which the contents of RI are to be placed.
- c) The stop digit must always be a 1. Thus the store order cannot be used as a stop order.
- d) At the start of the order the contents of the registers and x are:

R_1 b , the word to be stored
 R_2 irrelevant
 R^3 irrelevant
 x irrelevant

- e) At the end of the order the contents of the registers and x are:

R_1 b
 R_2 Unchanged
 R^3 0
 x b

0.14. THE STORE CLEAR ORDER.

- a), b), c) The same as for 0.13.
- d) At the start of the order the contents of the registers and x are:

R_1 Irrelevant

R_2 Irrelevant

R^3 Irrelevant

x Irrelevant

e) At the end of the order the contents of the registers and x are:

R_1 0

R_2 Unchanged

R^3 0

x 0

0.15. and 0.16. THE UNCONDITIONAL TRANSFER ORDERS.

a) The same as for 0.1.

b) The digits 0-9 (20-29) express the memory location to which the control is to be transferred. I.e., the location where the next order is to be found.

c) If the step digit is a 0, the control transfer takes place to the same phase of the new order-word as that of the order in question. If the step digit is a 1, the transfer takes place to the opposite phase.

d) At the start of the order the contents of the registers and x are:

R_1 Irrelevant

R_2 Irrelevant

R^3 Irrelevant

x b, the next order-word

e) At the end of the order the contents of the registers and x are:

R_1 Unchanged
 R_2 Unchanged
 R^3 0
 R_3 b
 x b.

0.17. and 0.18. THE CONDITIONAL TRANSFER ORDERS.

a), b), c) The same as for 0.15 - 16.

d) At the start of the order the contents of the registers and x are:

Case A

R_1 $a \geq 0$

R_2 Irrelevant

R^3 Irrelevant

x b, the next order-word

Case B

R_1 $a < 0$

R_2 Irrelevant

R^3 Irrelevant

x b, irrelevant

e) At the end of the order the contents of the registers and x are:

Case A

R_1 Unchanged

R_2 Unchanged

R^3 0

R_3 b

x b

Case B

R_1 Unchanged

R_2 Unchanged

R^3 b

R_3 Unchanged

x b

0.19. THE QUICK-SUM ORDER.

- a) The order may be only in the first phase of an order-word.
- b) The digits 0-9 express the memory location x from which the first operand is to come. It is obtained from the orders 0.1. - 0.8., inclusive, by setting digit 19 to 1. In this case the order specified without this digit being 1, i.e. one of the set 0.1 - 0.8, is performed first at the location x and then serially at each following location through 1023 after which the order terminates. The second phase order must be a transfer of the control to the location of the next order-word. This is due to the fact that the order counter no longer stores the location of the next order-word.
- c) The step digit must be a 1.
- d) At the start of the order the contents of the registers and x are:

R_1 a
 R_2 Irrelevant
 R^3 Irrelevant

$x + i$ b_i $i = 0, 1, \dots, 1023 - x$

- e) At the end of the order the contents of the registers and x are:

R_1 b_{1023} if 0.1, 3, 5 or 7
 $a + \sum f(b_{x+i})$ if 0.2, 4, 6, 8

R_2 Unchanged

R^3 b_{1023}

$x + i$ $b_i,$

where

$$f(b) = \begin{cases} b & \text{if } 0.2 \\ -b & \text{if } 0.4 \\ |b| & \text{if } 0.6 \\ -|b| & \text{if } 0.8 \end{cases}$$

0.20. RIGHT SHIFT, NO-ROUND OFF ORDER.

a) The same as for 0.1.

b) The digits 4-9 (24-29) express the number of shifts to be executed. This number n is expressed as an integer times 2^{-9} (2^{-29}). The digits 0 - 3 are irrelevant. The number stated in digits 4 - 9 (24-29) must not exceed 47 and must not be 0. (A shift by 0 is executed as a shift by 1.)

c) The same as for 0.1.

d) At the start of the order the contents of the registers are:

$$\begin{aligned} R_1 & a_0, a_1, a_2, \dots, a_{38}, a_{39} \\ R_2 & b_0, b_1, b_2, \dots, b_{38}, b_{39} \\ R^3 & \text{Irrelevant} \end{aligned}$$

e) At the end of the order for a shift of 1 the contents of the registers are:

$$\begin{aligned} R_1 & a_0, a_0, a_1, a_2, \dots, a_{38} \\ R_2 & a_{39}, b_0, b_1, b_2, \dots, b_{38} \\ R^3 & b_0, b_1, b_2, \dots, b_{39} \end{aligned}$$

for a shift of n this is iterated n times.

f) This order may be given with the clear digit, digit 18

(38), a 0 or a 1. The situation above shows the case of this digit = 0.

We show below the case when it is 1

e') At the end, for a shift of 1, the contents of the registers are:

$$\begin{array}{l} R_1 \quad a_0, 0, 0, 0, \dots, 0 \\ R_2 \quad 0, b_0, b_1, b_2, \dots, b_{38} \\ R^3 \quad b_0, b_1, b_2, \dots, b_{39} \end{array}$$

We note that even though R_1 has been cleared to 0 before the shift starts the sign of the number that was there is propagated by the shift.

0.21. RIGHT SHIFT, ROUND OFF ORDER.

This order is not yet available.

0.22. LEFT SHIFT ORDER.

a), b), c) The same as for 0.20.

d) At the start of the order the contents of the regis-

ters are:

$$\begin{array}{l} R_1 \quad a_0, a_1, a_2, \dots, a_{38}, a_{39} \\ R_2 \quad b_0, b_1, b_2, \dots, b_{38}, b_{39} \\ R^3 \quad \text{Irrelevant} \end{array}$$

e) At the end, for a shift of 1, the contents of the

registers are:

$$\begin{array}{l} R_1 \quad a_1, a_2, \dots, a_{38}, a_{39}, a_0 \\ R_2 \quad b_1, b_2, \dots, b_{38}, b_{39}, a_0 \\ R^3 \quad b_0, b_1, \dots, b_{38}, b_{39} \end{array}$$

For a shift of n this is iterated n times.

f) This order may be given with the clear digit a 0 or a 1.

The situation above shows the case of this digit = 0. We show below the case when it is 1.

e') At the end, for a shift of 1, the contents of the registers are:

$$R_1 \quad 0, 0, \dots, 0, 0, 0$$

$$R_2 \quad b_1, b_2, \dots, b_{38}, b_{39}, 0$$

$$R^3 \quad b_0, b_1, \dots, b_{37}, b_{38}, b_{39}$$

g) We indicate next what occurs when a left-shift (of 1) is followed by a right shift (of 1).

e") At the end of the order the contents of the registers are:

$$R_1 \quad a_0, a_1, a_2, \dots, a_{38}, a_{39}$$

$$R_2 \quad 0, b_1, b_2, \dots, b_{38}, b_{39}$$

$$R^3 \quad b_1, b_2, \dots, b_{39}, a_0$$

0.23. THE R_2 TO R_1 ORDER.

a) The same as for 0.1.

b) This order is actually not one but a set of eight different ones. These are essentially the orders 0.1. - 0.8. except that the operand comes not from a memory location x but rather from RII. One other feature is available in connection with this order. To describe this we consider the digits 0-9 (20-29). Of these 0 - 3 (20-23) are irrelevant. The digits 4 - 9 (24-29) are as indicated in b) of 0.20. They express an integer times 2^{-9} (2^{-29}) which is > 0 and ≤ 47 .

c) The same as for 0.1.

d) At the start of the order the contents of the registers are:

R_1 a
 R_2 b
 R^3 Irrelevant

e) At the end of the order the contents of the registers are:

$$R_1 \begin{cases} 2g(a) + (n+1) f(b) & \text{if } n \text{ is odd} \\ g(a) + nf(b)/2 & \text{if } n \text{ is even,} \end{cases}$$

where f is defined in the discussion of Order 19 and

$$g(a) = \begin{cases} a & \text{if } 0.2, 4, 6, 8 \\ 0 & \text{if } 0.1, 3, 5, 7 \end{cases}$$

R_2 b
 R^3 b.

0.24. and 0.25. IBM AND DRUM PRIMING ORDER.

a) These orders must be second phase ones. Their use is to specify the number of cards to be read or punched by the IBM reproducer or the starting word block number and the number of such blocks to be loaded or unloaded on or from the drum.

There are 12 words on an IBM card occupying the columns 1-4, 6-9, 11-14, 16-19, 21-24, 26-29, 31-34, 36-39, 41-44, 46-49. The columns 5, 10, 15, 20, 25, 30, 35, 40, 45 are never used. Columns 66-80 are used for identifying information for the human operator. Each word occupies a row on the card, a hole indicating a 1, no hole a 0.

The Magnetic Drum contains 2048 words divided into two main groups of 1024 each. Each of these is composed of 32 blocks of 32 words each.

b) The address portion of the order is used to specify the number of cards in the case of IBM operation or the starting block number and the number of blocks in the case of drum operation. In the former case, the IBM one, the digits 20-22 are irrelevant; 23-29 express as an integer times 2^{-29} the number of cards to be processed. In the latter case, the drum one, the digits 20-24 are used to express the starting block number as an integer times 2^{-24} ; the digits 25-29 the number of blocks to be processed, the number being an integer times 2^{-29} . It is important to note that the blocks and the number of blocks are counted 1, 2, ..., 32. Thus 00000 means block 32 if it appears in positions 20-24 and means 32 blocks if in 25-29.

c) It is best always to put a step-digit, digit 30, into this order.

In principle this is not one but a set of 18 orders, 0.1 - 0.18; it differs from these only in that the address portion does two things: It is not only the operand for the order 0.1 - 0.18 but is also sent to a special register in the IBM-Drum Control. It remains there permanently until altered by another such order.

Since the analogous 0.1 - 0.18 order will be executed using the given address as operand it will bring an irrelevant quantity into the Arithmetic Unit. Thus if the contents of RI are relevant and it is desired to prime, it is best to use a load R_2 type of order, whereas if the contents of RII are relevant, one of the type 0.1 - 0.8. The priming order differs from 0.1 - 0.18 only in that digit 31 is a 0.

0.26. IBM INPUT TO MEMORY ORDER.

a) This must be first phase. The second phase of the same order-word must be a transfer of the control to the next order-word.

b) The digits 0-9 are used to specify the memory location x for which the order is first executed. The order is then executed for $x + 1$, $x + 2$, ... until the number of cards previously set by the prior priming order has been reached.

c) The step digit must be a 1.

d) The only register that is unchanged is RII.

0.27. IBM OUTPUT TO MEMORY ORDER.

a), b), c) The same as for 0.26.

d) All registers are altered.

0.28. DRUM INPUT TO MEMORY ORDER.

a) The same as for 0.26.

b) The digits 0-9 are used to specify the memory location x for which the order is first executed. The order is then executed for $x + 1$, $x + 2$, ... until the number of blocks previously set by the prior priming order has been reached.

c) The same as for 0.26.

d) The same as for 0.26.

e) The drum stores 1024 bits per track and has 80 tracks.

These 80 are divided into 2 sets of 40 each which constitute the major groups mentioned above in the discussion of order 24. The selection of the proper group is controlled by digit 15. If it is a 0, group A is selected; if it is a 1, group B is selected.

0.29. DRUM OUTPUT FROM MEMORY ORDER.

a), b), c) The same as for 0.28.

d) The same as for 0.27.

e) The same as for 0.28.

digital representation of

ORDERS

41.

0	No Step	Ext.	Arith.	AT	#	+R	x/÷	No RO	Hold	Spare
1	Step	Int.	Wms.	NAT	Abs.	-L	Σ	RO	Clear	

ADDRESS	ORDER	CODE FORM	NOTES
---------	-------	-----------	-------

Summation

1.	+ clear	1/0 1 1 1 0 0 1 0 1 0	1.
2.	+ hold	1/0 1 1 1 0 0 1 0 0 0	
3.	- clear	1/0 1 1 1 0 1 1 0 1 0	
4.	- hold	1/0 1 1 1 0 1 1 0 0 0	
5.	+ abs. clear	1/0 1 1 1 1 0 1 0 1 0	
6.	+ abs. hold	1/0 1 1 1 1 0 1 0 0 0	
7.	- abs. clear	1/0 1 1 1 1 1 1 1 0 1	
8.	- abs. hold	1/0 1 1 1 1 1 1 1 0 0	

Memory
orders

(address
specifies word)
in memory)

0-1023

Other NAT Orders

9.	x clear NRO	1/0 1 1 1 0 0 0 0 1/0 0	2.
10.	x clear RO	1/0 1 1 1 0 0 0 1 1/0 0	
11.	⋆	1/0 1 1 1 1 1 0 0 0 0	
12.	Load R ₂	1/0 1 1 1 0 0 1 1 0 0	

Arith. Trivial Orders

13.	Store	1 1 1 0 1 0 1 0 0 0	3.
14.	Store Clear	1 1 1 0 1 0 1 0 1 0	
15.	Un. T. C. no step	0 1 1 0 1 0 0 0 0 0	4.
16.	Un. T. C. step	1 1 1 0 1 0 0 0 0 0	
17.	Con. T. C. no step	0 1 1 0 1 1 0 0 0 0	
18.	Con. T. C. step	1 1 1 0 1 1 0 0 0 0	

Special Order

19.	Quick Sum.	1 1 1 1 1/0 1/0 1 0 1/0 1	5.
-----	------------	---------------------------	----

Non Memory Orders

20.	Sh. Right NRO	1/0 1 0 1 0 0 0 0 1/0 0	
21.	Sh. Right RO	1/0 1 0 1 0 0 0 1 1/0 0	6.
22.	Sh. Left	1/0 1 0 1 0 1 0 0 1/0 0	
23.	R ₂ --> R ₁	1/0 1 0 1 1/0 1/0 1 0 1/0 0	7.

Input-Output Orders

24.	IBM priming	1 0 1 1/0 1/0 1/0 1/0 1/0 1/0 0	8.
25.	Drum priming	1 0 1 1/0 1/0 1/0 1/0 1/0 1/0 0	9.
26.	IBM Load	1 0 0 0 0 0 1 0 0 0	10.
27.	IBM Punch	1 0 0 1 0 0 1 1 1 0	
28.	Drum --> Memory	1 0 1 0 1/0 1 0 0 0 0	11.
29.	Memory --> Drum	1 0 1 1 0 1/0 1 1 1 0	11.

1-127

0-31; 0-31

0-1023

1-47

1-47

1-47

2